User's Manual

Inform Your Process

Microsoft, Windows, Windows NT, Windows 2000, Windows XP, Visual Basic, Microsoft .NET, Visual C++, Visual C#, and ActiveX are either trademarks or registered trademarks of Microsoft Corporation.

All other nationally and internationally recognized trademarks and tradenames are hereby recognized.

Copyright © 2002-2008 by MVTec Software GmbH, München, Germany

Edition 1	November 2002	(ActivVisionTools 2.1)
Edition 2	January 2005	(ActivVisionTools 3.0)
Edition 3	February 2006	(ActivVisionTools 3.1)
Edition 4	May 2008	(ActivVisionTools 3.2)

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

More information about ActivVisionTools can be found at:

`http://www.activ-vision-tools.com`

How to Read This Manual

This manual explains how to output the results of other ActivVisionTools via the serial interface (RS-232) using ActivSerial. It describes the functionality of ActivSerial and its cooperation with other ActivVisionTools with Visual Basic examples. Before reading this manual, we recommend to read the manual Getting Started with ActivVisionTools, which introduces the basic concepts of ActivVisionTools, and the User's Manual for ActivView to learn how to load and display images.

For each example in this manual, there is a corresponding Visual Basic project; these projects can be found in the subdirectory `examples\manuals\activserial` of the ActivVisionTools base directory you selected during the installation (default: `C:\Program Files\MVTec\ActivVisionTools`). Of course, you can also create your own Visual Basic projects from scratch. Note that the screenshots in the manual may differ slightly from the corresponding Visual Basic projects. To follow the examples actively, first install and configure ActivVisionTools as described in the manual Getting Started with ActivVisionTools.

We recommend to **create a private copy of the example projects** because by experimenting with the projects, you also change their *state*, which is then automatically stored in the so-called description files (extension `.dsc`) by ActivVisionTools. Of course, you can restore the state of a project by retrieving the corresponding description file from the CD.

Contents

1 About ActivSerial **1**

1.1 Introducing ActivSerial 2

1.2 The Sub-Tools of ActivSerial 2

2 Using ActivSerial **5**

2.1 Configuring the Communication via the Serial Interface 6

2.2 Selecting the Output Data 8

3 Tips & Tricks **11**

3.1 Customized Output Via the Programming Interface 12

Chapter 1

About ActivSerial

This chapter will introduce you to the features and the basic concepts of ActivSerial. It gives an overview about ActivSerial's *master tool* and its dialogs, which are described in more detail in chapter 2 on page 5.

1.1 Introducing ActivSerial **2**
1.2 The Sub-Tools of ActivSerial **2**

1.1 Introducing ActivSerial

In many machine vision applications, the task is not only to inspect images and calculate certain results but to communicate those results to the surrounding process. The process can then react on them, e.g., remove a faulty part from the conveyor belt. One possible communication channel between the process and the computer is the *RS-232 serial interface*. This interface was originally developed to use terminals with remote computer systems, therefore most PCs already come with one or more of such ports. On the other side, many (stored-program) control systems are also equipped with a serial interface, primarily to download programs from the host computer.

As its name suggests, the RS-232 serial interface transmits data as a series of bits, which are partitioned into groups of 5-8 bits and enclosed by special start and stop bits, and perhaps also a parity bit for error checking. Typically, those bit groups are interpreted as characters. Furthermore, the RS-232 protocol provides different mechanisms for data flow control by software or by hardware.

ActivSerial provides an easy-to-use access to the RS-232 serial interface. All you need to to it to configure the communication and to select which results and/or evaluations are to be transmitted.

Please note that **ActivSerial cannot be used to input data** via the serial interface!

An alternative method to connect to your process control is to use a so-called *digital I/O board*. These boards provide multiple input and output lines to signal events to and from the process. For more information please refer to the User's Manual for ActivDigitalIO.

1.2 The Sub-Tools of ActivSerial

ActivSerial provides only a *master tool*, `AVTOutputSerial`, and two dialogs that let you select results for output and configure the output format. In figure 1.1 `AVTOutputSerial` and the two dialogs are shown in an example ActivVisionTools application, where ActivSerial used to output the results of ActivBarcode.

`AVTOutputSerial` is the *master tool* of ActivSerial. It allows to enable the output and to open the two dialogs described below. How to use `AVTOutputSerial` is described in more detail in chapter 2 on page 5.

Via `AVTOutputSerial`, you can open a dialog that allows to configure the communication via the serial interface and the output format. How to use this dialog is described in section 2.1 on page 6.

A second dialog allows to select which results of which ActivVisionTool are to be sent via the serial interface. How to use this dialog is described in section 2.2 on page 8.

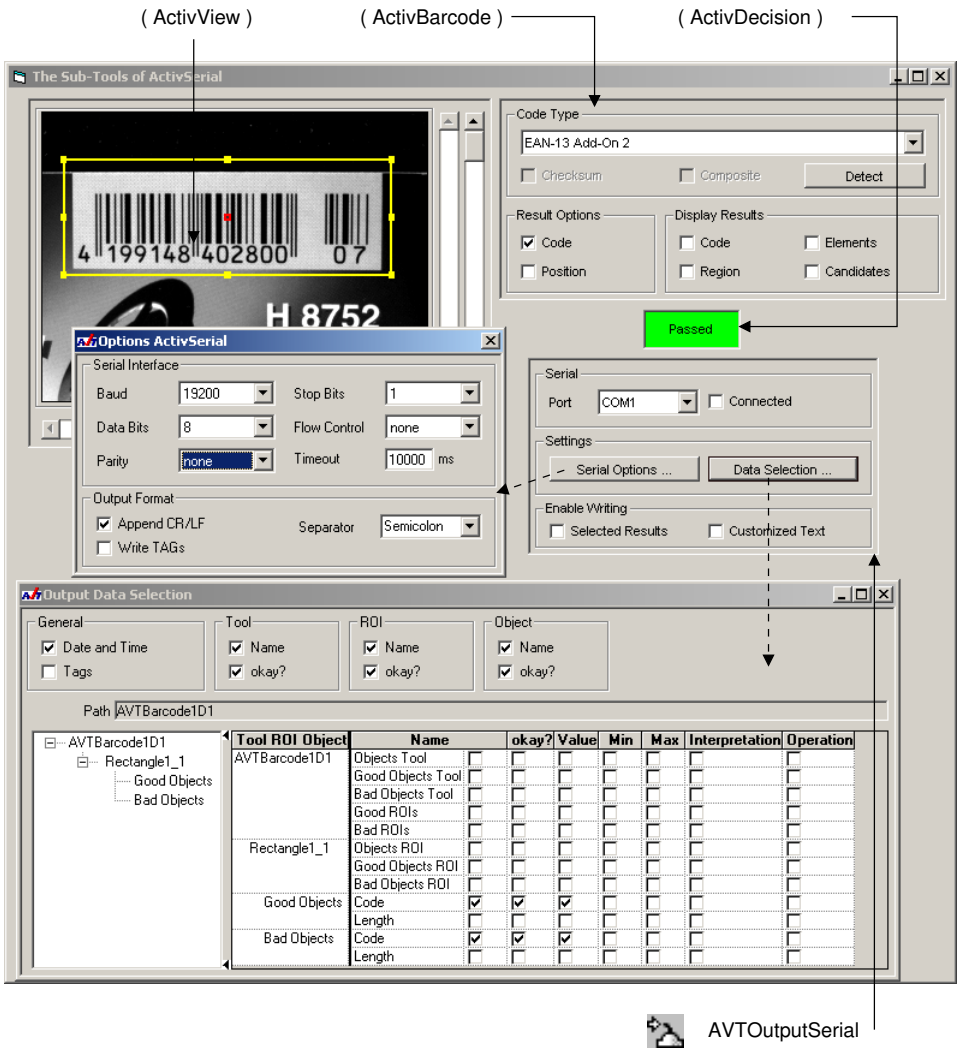

Figure 1.1: The sub-tools of ActivSerial in a bar code inspection application.

Chapter 2

Using ActivSerial

This chapter will explain how to use ActivSerial to output the results and evaluations of other ActivVisionTools via a serial interface.

In the corresponding Visual Basic projects, ActivBarcode is used as the tool whose results are to be transmitted. In fact, the projects already contain a completely configured application which checks the bar code printed on magazines. If you want to experiment with the projects in more detail we recommend to read the User's Manual for ActivBarcode and the User's Manual for ActivDecision.

2.1 Configuring the Communication via the Serial Interface 6
2.2 Selecting the Output Data 8

2.1 Configuring the Communication via the Serial Interface

The RS-232 serial protocol leaves some room for configuration. For example, you can transmit data at different speeds (rates) or choose to include a parity bit after each character for error checking. These parameters can be specified in the dialog `DialogSerialOptions`.

Of course, the selected values must be identical to the ones selected "on the other side of the line", i.e., for the device that reads data from the serial interface.

Visual Basic Example

Preparation for the following example:

- ☐ Please open the project `configuring\serial_configuring.vbp`, which contains an already configured bar code inspection application.
- ☐ Execute the application (`Run` ▷ `Start` or via the corresponding button). Load the image sequence `barcode\magazine1.seq`.

The following steps are visualized in figure 2.1.

1. The combo box `Port` allows to select the serial interface port which is to be used for output. `AVTOutputSerial` automatically opens the corresponding device. If it succeeds, the check box to the right is checked; if the device cannot be opened, e.g., because another application already uses it, an error message appears. You can uncheck the box to close the connection.
2. Open `DialogSerialOptions` by clicking `Serial Options` in `AVTOutputSerial`.
3. Specify the data rate used for the transmission in the combo box `Baud`.
4. The number of bits per character can be selected in the combo box `Data Bits`.
5. The combo box `Parity` offers different modes for error checking.
6. The number of stop bits transmitted after each character can be selected in the combo box `Stop Bits`.
7. The combo box `Flow Control` offers different modes for data flow control.
8. In the text box `Timeout` you can specify a timeout for the communication.
9. The elements in the frame `Output Format` allow to configure the format of the output data itself. They are described in the following section.

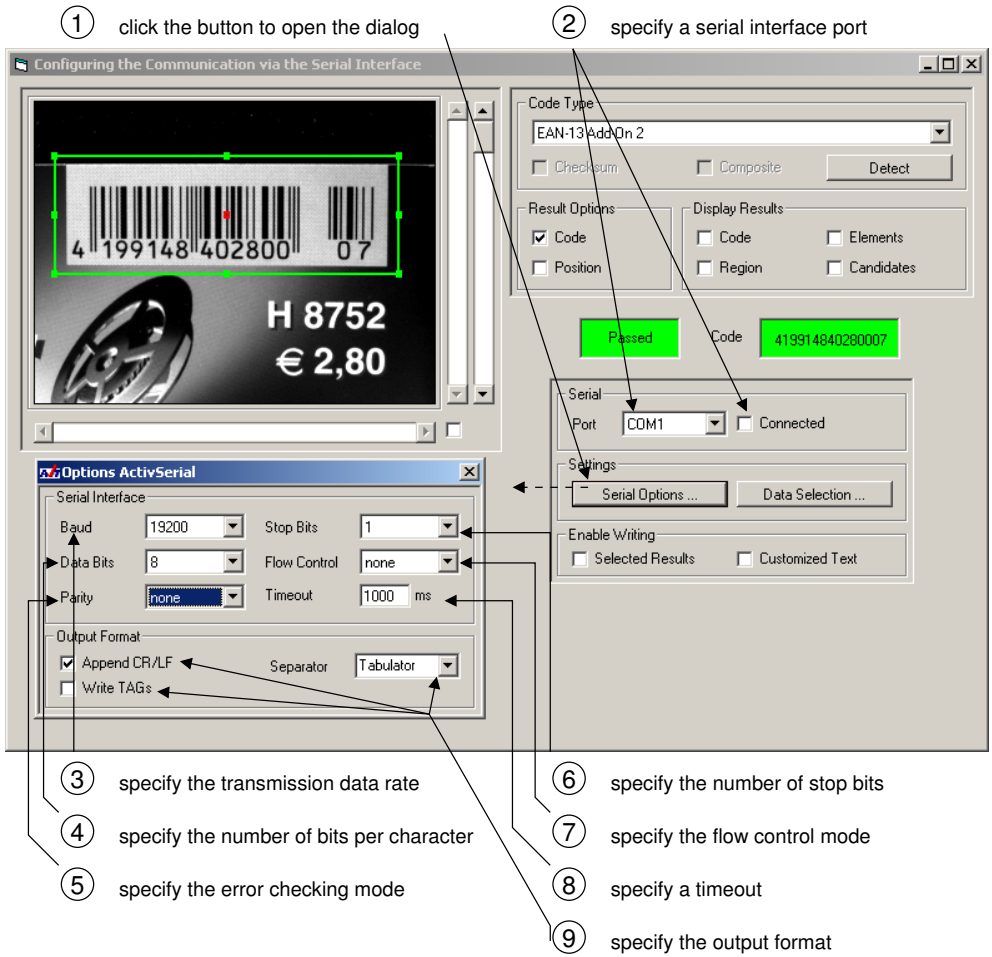

Figure 2.1: Configuring the communication protocol.

If your computer is equipped with two serial interface ports, you can test the communication by connecting the two ports with a suitable cable (crossed input/output lines) and then using a program called HyperTerminal, which is part of most Windows distributions, to read from the serial interface. The HyperTerminal file `COM2forAVT.ht` in the subdirectory `examples\manuals\activserial` of the ActivVisionTools base directory is configured to match the example projects; double-click it in the Windows Explorer to start it. You can check the values of the communication parameters by opening the dialog `File` ▷ `Properties` and then clicking `Configure` (after closing the connection).

2.2 Selecting the Output Data

The next step after configuring the communication is to specify what is to be transmitted using the dialog `DialogOutputDataSelect`. In this dialog, you can select from all available results, i.e., the calculated features of objects, ROIs, and tools and, if you are using ActivDecision as in the example projects, the evaluation of features, objects, ROIs, and tools.

Section 3.1 shows how to further customize the output via the programming interface.

Visual Basic Example

Preparation for the following example:

- ☐ If you worked on the previous example, you may continue using this project. Otherwise, open the project `selecting\serial_selecting.vbp`.
- ☐ Execute the application and load the image sequence `barcode\magazine1.seq`.

The following steps are visualized in figure 2.2 (only components of ActivSerial shown).

1. Click `Serial Options` and `Data Selection` in `AVTOutputSerial` to open the dialogs `DialogSerialOptions` and `DialogOutputDataSelect`.
2. In the left part of `DialogOutputDataSelect`, you can navigate through the results.
3. In the right part of `DialogOutputDataSelect`, you can select data for output by checking the corresponding boxes. You may output different items depending on the evaluation of an object. By clicking on the column labels with the right mouse button, you can check or uncheck all boxes in this column; similarly, you can check and uncheck whole rows or all rows of a certain tool.

   In the frame `Output`, you can select whether to transmit the date and time of the current processing cycle in front of the selected results in the check box ☑ `Date and Time`. If you check ☑ `Tags`, the data is enclosed by tags. Note that both options are more useful in conjunction with ActivFile, which also uses this dialog.
4. In the frame `Output Format` of `AVTOutputSerial`, you can specify the output format of the individual data items. If you check ☑ `Append CR/LF`, a carriage return and linefeed character is transmitted after each line. Note that a line can consist of multiple items, which are separated by the delimiter selected in the combo box `Separator`. For example, all items selected for a certain feature are transmitted in one line.
5. If you check ☑ `Selected Results` in `AVTOutputSerial`, the output is enabled; the use of the other check box is explained in section 3.1 on page 12. At the end of the next processing cycle, the selected data is transmitted via the serial interface. The figure shows an example output on the HyperTerminal; as described in the previous section, a

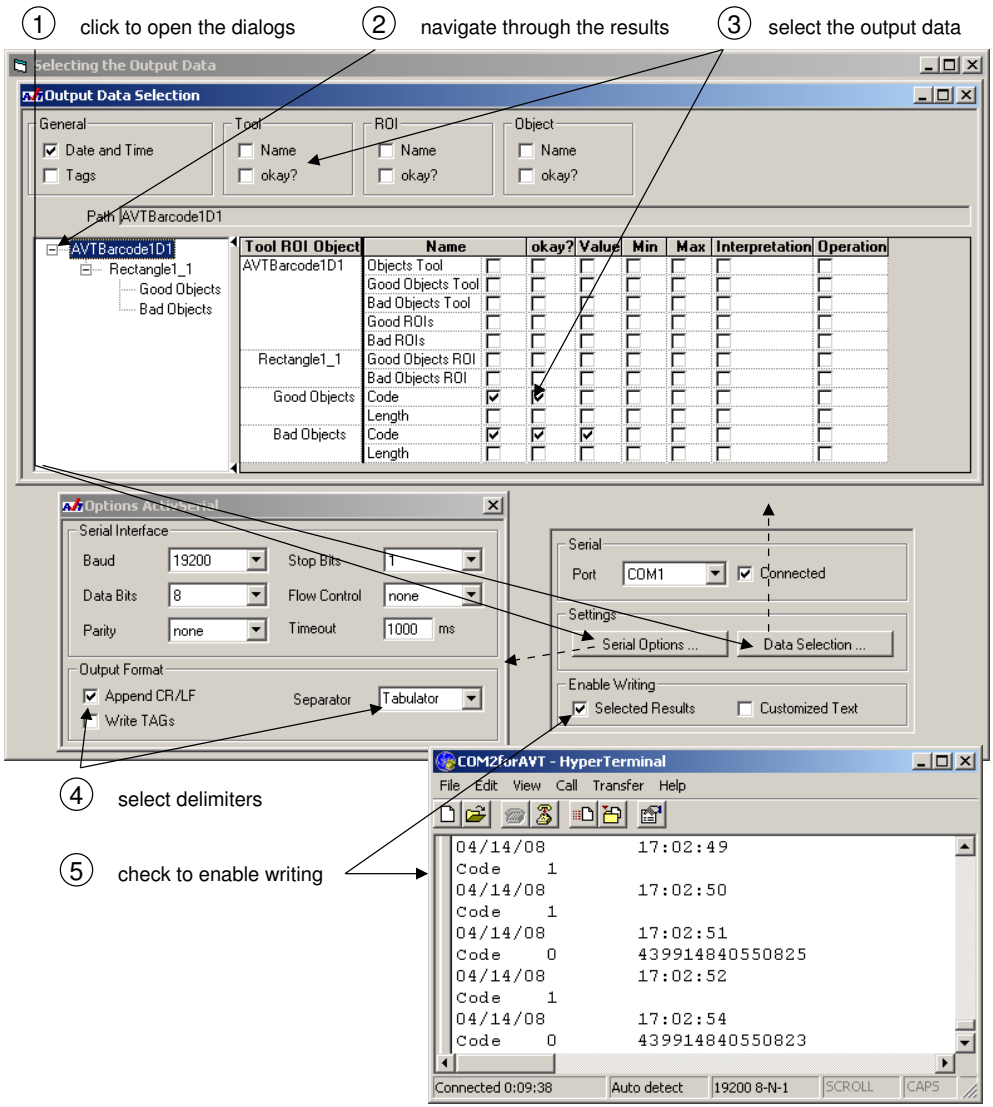

Figure 2.2: Selecting the output data.

preconfigured version for the use with the example projects can be found in the in the subdirectory `examples\manuals\activserial`.

Chapter 3

Tips & Tricks

This chapter contains additional information about ActivSerial, e.g., how to use the programming interface of ActivVisionTools to output customized text via the serial interface.

3.1 Customized Output Via the Programming Interface 12
3.1.1 Basic Principles 12
3.1.2 Example 13

3.1 Customized Output Via the Programming Interface

The previous chapters and sections showed how to use the ActivSerial interactively, i.e., via the graphical user interfaces presented by the underlying ActiveX controls. In this mode, you can configure the output via the serial interface rapidly and easily, without any programming. However, there is more to ActivVisionTools than the graphical user interfaces: Because ActivVisionTools comes as a set of ActiveX controls, it provides you with an open programming interface, thereby offering full flexibility.

In this section, we show how to output customized text via the programming interface. Please note that we assume that readers of this part have at least a basic knowledge of Visual Basic.

! To work with the programming interface, you must first **add the ActivVisionTools type library to the project's references** by checking the box labeled `ActivVisionTools Type Library` in the Visual Basic menu dialog `Project` ▷ `References`.

Detailed information about the programming interface can be found in the **Reference Manual**.

3.1.1 Basic Principles

The programming interface ActivSerial provides four events, which allow to integrate your own code into the corresponding event handlers; below, we describe the three that are most useful for customixing the output:

- `Start`

  This event is raised when ActivSerial starts to execute. In a parameter, it passes the results of the connected ActivVisionTools, i.e., the calculated features values and their evaluations. This access to the results is important to determine the content of your customized output.

  A corresponding event handler looks like this:

```
Private Sub AVTOutputSerial1_Start(atIconicData As _
                                   ActivVTools.AVTImageMessageClass, _
                                   cToolResults As Collection)
```

- `WriteSelectedResults`

  If the box ☑ `Selected Results` in `AVTOutputSerial` is checked, this event is raised before ActivSerial transmits the result data selected in the dialog `DialogOutputDataSelect`. The event has a single parameter, which contains the formatted output text. This parameter can be modified in a corresponding event handler.

```
Private Sub AVTOutputSerial1_WriteSelectedResults(sText As String)
```

  To be more exact, the event `WriteSelectedResults` is raised as soon as the internal

output buffer is full. If many results are selected, the event may therefore be raised more than once for one processing cycle. In the example described in the next section, the event is therefore only used to prevent the output of the selected results in a certain execution mode.

- `WriteCustomizedText`

  As its name suggests, this event is the main gate to customized output. It is raised if the box ☑ `Customized Text` in `AVTOutputSerial` is checked. As a parameter, the event passes an (empty) string:

  ```
  Private Sub AVTOutputSerial1_WriteCustomizedText(sCustomizedText As String)
  ```

  In the event handler, you can insert your own data into the string, which is then automatically transmitted once per cycle.

3.1.2 Example

The ActivVisionTools distribution includes the example Visual Basic project `customized\serial_customized.vbp`, which shows how output customized text. The following tasks are to be solved via the programming interface:

- In the *configuration mode*, selected results as shown in figure 2.2 on page 9 are to be transmitted.
- In the *application mode*, customized data is to be transmitted, consisting of an extended evaluation (`'0'`: incorrect code; `'1'`: correct code), followed by the last two digits of the code, which encode the issue of the magazine (only if code is evaluated as 'okay'). Figure 3.1 shows this customized output for the complete image sequence.

The example project is already configured; just start it and click the button `Run` in `AVTViewExecute`. As described in section 2.1 on page 6, the preconfigured HyperTerminal `COM2forAVT.ht` for the use with the example projects can be found in the in the subdirectory `examples\manuals\activserial`.

In the following, we take a look at the Visual Basic code that solves the tasks described above. The first problem is that we need the actual results of the processing cycle, i.e., the extracted bar code and its evaluation, when creating the customized output in the handler for the event `WriteCustomizedText`, but this information can only be accessed within the handler for the event `Start`. Therefore, the following variables are declared globally:

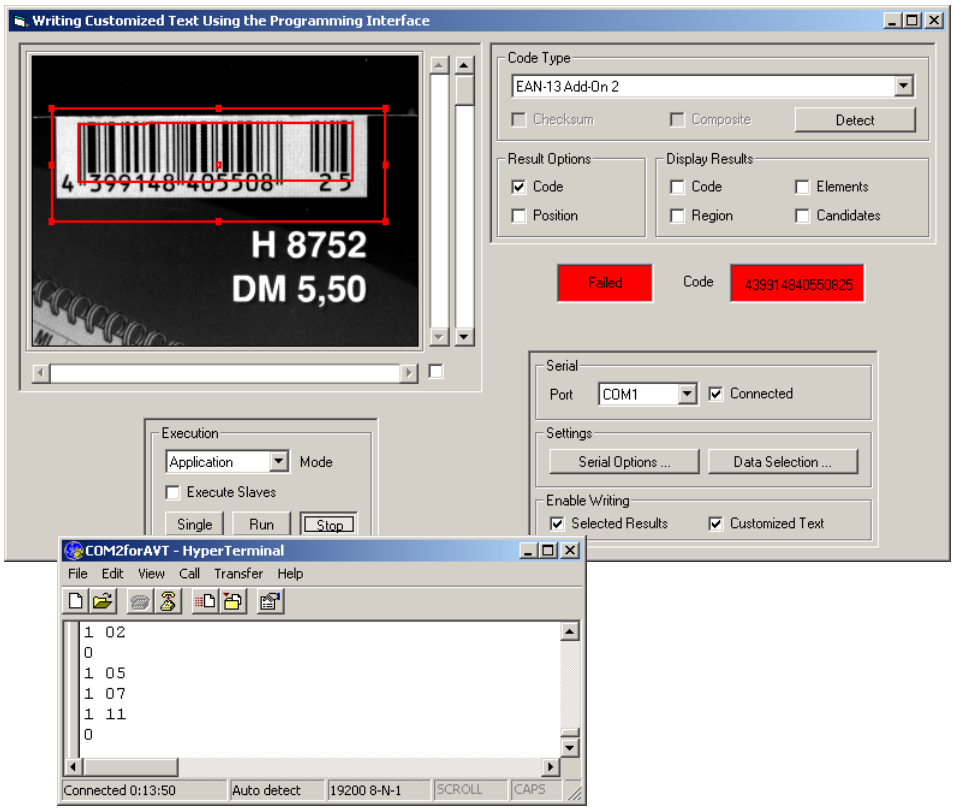

Figure 3.1: Customizing the output text.

```
Private bBarcodeResultsExist As Boolean
Private sCode As String
Private bCodeEval As Boolean
Private handleCode As Integer
```

The following code “fills” these variables with results in the handler for the event `Start`. First, the results of ActivBarcode are extracted from the collection of tool results:

```
bBarcodeResultsExist = False
For Each atToolResult In cToolResults
  If atToolResult.Name = "AVTBarcode1D1" Then
    Set atBarcodeResult = atToolResult
    bBarcodeResultsExist = True
  End If
Next
If bBarcodeResultsExist = False Then
  Exit Sub
End If
```

Then, we focus on the results of the first ROI:

```
Set atROIResult = _
  atBarcodeResult.ROIResult(atBarcodeResult.ROIResultNames(0))
```

The relevant evaluations and feature values all belong to the first object in the ROI:

```
bCodeEval = atROIResult.ObjFeatureEvaluation(handleCode, 0)
sCode = atROIResult.ObjectValue(handleCode, 0)
```

The variables for the feature handles are also declared globally and set upon loading the form:

```
Private handleCode As Integer

Private Sub Form_Load()
  handleCode = AVTBarcode1D1.FeatureHandleCode
End Sub
```

For more details about the access of feature values and evaluations, please refer, e.g., to the User's Manual for ActivBarcode, section 4.3 on page 26.

In the handler for the event `WriteCustomizedText`, the customized output can now be created based on the global variables containing the extracted code and its evaluation:

```
Private Sub AVTOutputSerial1_WriteCustomizedText(sCustomizedText As String)

  Dim sOutputText As String, sMonth As String

  If bCodeEval = False Then
    sOutputText = "0"
  Else
    sMonth = Right$(sCode, 2)
    sOutputText = "1" & " " & sMonth
  End If

  sCustomizedText = sOutputText & vbCrLf

End Sub
```

The customized output is restricted to the *application mode* by inserting the following code at the beginning of the event handler.

```
If Not AVTView1.ExecutionMode = eApplicationMode Or _
    bBarcodeResultsExist = False Then
  Exit Sub
End If
```

On the other side, the output of the selected results is restricted to the *configuration mode* by removing the output data from the string parameter in the handler for the event `WriteSelectedResults` in the *application mode*:

```
Private Sub AVTOutputSerial1_WriteSelectedResults(sText As String)
  If AVTView1.ExecutionMode = eApplicationMode Then
    sText = ""
  End If
End Sub
```

When using the programming interface of ActivVisionTools, you leave the safe world of the graphical user interfaces where all input is checked for validity automatically. In contrast, if you try to access a non-existing object or result via the programming interface, a run-time error is caused which terminates your application! To avoid this, you can use the Visual Basic error handling mechanisms, i.e., set up an error handler which examines any occurring error and reacts in a suitable way. For example, the event handler vor `Start` contains the following error handler:

```
Private Sub AVTOutputSerial1_Start(atIconicData As _
                                   ActivVTools.AVTImageMessageClass, _
  ' variable declarations
On Error GoTo ErrorHandler
  ' body of the procedure
  Exit Sub

ErrorHandler:
  Dim sTitle As String
  If Left(Err.Source, 11) = "ActivVTools" Then
    sTitle = "ActivVisionTools Error"
  ElseIf Left(Err.Source, 10) = "Customized" Then
    sTitle = "Incorrect Project Configuration"
  Else
    sTitle = "Runtime Error " & CStr(Err.Number)
  End If
  Call MsgBox(Err.Description, vbExclamation, sTitle)
  AVTView1.RunState = False
End Sub
```


The main goal of the error handler is to react upon an ActivVisionTools error with a popup dialog instead of terminating the application. For this, the source of the error is examined. Within the example project, the correct configuration of the project is then checked. For example, the following code checks whether the bar code is extracted at all, i.e., whether the box ☑ Code in AVTOutputSerial is checked; if not, an error with the source set to "Customized" is raised (and then "caught" in the error handler):

```
If atBarcodeResult.FeatureUsed(handleCode) = False Then
   Call Err.Raise(1, "Customized", "Code must be calculated!")
End If
```

To test the effect, click Run and then uncheck ☑ Code; the dialog shown in figure 3.2a appears.

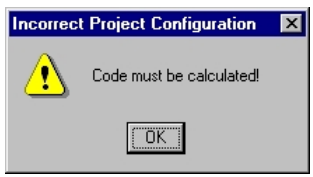

Figure 3.2: Popup dialog resulting from not extracting the bar code.

The following code checks that exactly one ROI is present, in which at least one bar code is extracted, before accessing the results. To test the effect, start the continuous execution and create a second ROI or select another bar code type in `AVTBarcode1D`.

```
End If
  ' focus on ROI
  If atROIResult.ObjectNum > 0 Then
    ' access results
  Else
    Call Err.Raise(1, "Customized", "No bar code found!")
  End If
Else
  Call Err.Raise(1, "Customized", "Expecting exactly 1 ROI!")
End If
```